Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

Wu Xiaobo

Efficient Implementation of Coreset-based K-Means Methods

Master's Thesis
Espoo, May 14, 2021

Supervisor: Professor Jiang Shaofeng, Aalto University
Advisor: Jiang Shaofeng

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

ABSTRACT OF MASTER'S THESIS

Author: Wu Xiaobo

Title:
Efficient Implementation of Coreset-based K-Means Methods

Date: May 14, 2021 **Pages:** 50

Major: Machine Learning, Data Science and Artificial Intelligence **Code:** SCI3044

Supervisor: Professor Jiang Shaofeng

Advisor: Jiang Shaofeng

The problem of efficiently extracting information from big data has received significant attention in both academia and industry. In the last decades, many computational models like distributed/parallel computing, streaming algorithms and dynamic algorithms have been proposed to cope with big data. These methods were commonly used nowadays and have obtained great performances in many scenarios related to dealing with a massive amount of data. However, they are algorithm-driven approaches which means you need to generate new ideas every time, and it still will be constrained by the growing data size.

Recently, a data-driven approach named coreset was developed to generate a much smaller summary to the original big data and directly apply the existing algorithms in the small dataset to reach an acceptable result in a very decent time and space complexity. In the latest research, the size of coreset $|S|$ can be handle to uncorrelated with the original data size $|D|$, which means unlimited by the size of data. My main result is an efficient implementation of an open-source coreset-based modern computational framework based on C++ and CUDA programming language in this thesis project.

Specifically, we implement a coreset library that can construct a small coreset for arbitrary shapes of numerical data with a decent time cost. My job was mainly to follow the research proposed by Braverman et al. (SODA 2021). The experiments were conducted to benchmark with the CuML library, a well-known collection of the efficient GPU implementation of various machine learning algorithms. The benchmark results prove our implementation of the coreset method can lead to a faster speed in solving the k-means problem among the big data with high accuracy in the meantime. In addition, by utilizing the composable property of the coreset, our merge-and-reduce version of the coreset breaks through the restriction of GPU memory.

Keywords: coreset, k-means, cuda, c++, cuml, machine learning, parallel

Language: English

Acknowledgements

First of all, I want to express my appreciation to my supervisor and advisor, Prof. Jiang Shaofeng. Without his helpful comments, seriously guidance, far-sighted suggestion, my thesis project could not progress smoothly and lead to the current stage. Moreover, I want to thank Prof. Jiang Shaofeng for holding weekly discussion meetings with me and answering my questions in online chat and email quickly. Whenever I got stuck in some tricky problems, the guidance from Prof. Jiang Shaofeng would always make me clear about what is the next thing that I should do. Also, I would like to appreciate the staffs in the computer science department who assisted me throughout the procedures of my thesis working. Besides, I would like to thanks my friends who cure my mood when I am in a stressful condition during the last six months of working and studying. Finally, I want to thank my loved ones and my families, who gave me comfort and company when I am low throughout the entire process of my thesis. It will be much harder for me to persist on this long researching project without their loves.

Thank you, and wish all the best!

Espoo, May 14, 2021

Wu Xiaobo

Abbreviations and Acronyms

CuML	CuML is a well-known collection of the efficient GPU implementation of various machine learning algorithms
CUDA	Compute Unified Device Architecture (CUDA), is a modern platform for GPU computing that proposed by Nvidia company.
SparkML	Machine learning APIs collections that proposed as one part of the Spark
MATLAB	abbreviation of "matrix laboratory"), a well-used programming language for multi-paradigm and numeric computing.
OpenCV	(Open Source Computer Vision Library), is a state of the art framework in computer vision area.
R language	is a well-known programming language in computing science and mathematical area.
Scikit-learn	is a open-source Python library offering efficiently methods in solving machine learning problems.
Vim	is a famous text editor for Unix-like system that first proposed by Bill Joy's.
IDE	(integrated development environment), is an application that offers comprehensive facilities for program developing.
CSV	(comma-separated values), is a text file format that regard a comma as separation between meaningful values.

Contents

Chapter 1

Introduction

With the rapidly growing trend in the usage rate of internet and mobile devices, roughly 2.5 exabytes of data ($2.5 \cdot 10^{18}$) [8] is created every day by many affordable electric devices like mobile phones, laptops, computer servers, smartwatches, cameras, smart machines and so on [17, 23]. Coresets are present-day means for efficiently analysing big data that have been raised considerable attention in theoretical computer science, networking, machine learning and many other fields. The general concept of coresets [42] is to produce a small subset S of big data set D, and let $0 < \epsilon$, f be a measurable function, for $\forall c$ in Solutions class C, it has:

$$|f(D, c) - f(S, c)| \leq \epsilon \cdot f(D, c)$$

Currently, coresets have been researched for many fundamental problems in machine learning and computer science areas, including clustering [4], classification [20], and regression [10]. In the past twenty years, many researchers proposed a bunch of widely-used methods to solve these problems. For instance, the k-means [32] approach is an efficient algorithm with a simple workflow to solve the clustering problem, and it has been applied widely in industry and proved to obtain very decent performance in many practical scenarios. Even in dealing with a relatively big data set, with the appropriate programming level optimization and well-performed hardware, the K-means algorithm will still lead to an efficient processing procedure. However, with the rapid popularity of the Internet and mobile phones in last two decades, the amount of data that computer algorithms need to face increases exponentially. Namely, we are currently in the era of big data, which refers to high-growing data sets that contain multiple forms: structured, unstructured and semi-structured data [47]. Furthermore, We all know that there is a well-known Moore's Law in the computer field, that assert the number of transistors on a CPU will doubles in every two years and the cost

of computers will halve. This famous law indicates a very significant opinion that it is possible to expect the performance and capability of common-used computers or intelligent machines improving in every couple of years, and even with a less spend. In other words, higher performance computer hardware will be available with a less cost in every two years. However, Thomas N. proposed a different idea in 2017 indicate that Moore's Law is already lose correctness, the termination of Moore's Law and a new starting point for computer science and information technology is coming [59]. Those phenomena mean that our hardware and traditional algorithms have some limitations in processing more massive data.

Coreset is one of the possible solutions that utilizing the data reduction conception to dealing with big data problem, and it can be corporate with any other traditional algorithms with means it is not an algorithm-driven method. Another significant property of coreset is composable which means: the union of coreset(X) and coreset(Y) is a coreset for ($X \cup Y$). Based on composability of coreset, coreset constructions in classical setting can be transformed to streaming [22], distributed [5], and dynamic [24] model easily by utilizing the idea of merge-and-reduce. This attribute greatly simplify the construction of coresets and flexibly constructed a coreset of astronomical data quickly. However, the development of coreset is still limited to the academic area. Popular open-source libraries like SparkML and CuML do not employ coresets to implement approaches for machine learning problems. Gaps between industry and academia are shown. Therefore, the main contribution of my thesis works is to fill in the black zone between theory and practice of coreset-based approaches.

1.1 Problem Statement

The theoretical research of coresets have been analysed by many researchers in the past decade [2, 12, 34, 40, 42, 54]. Coreset based methods have been studied in many artificial intelligence and machine learning areas and show an outstanding result. Nevertheless, there are few publications or open-source libraries related to coreset that has been proposed, which lead to a heavy pipeline for the researchers or students in other areas to construct coreset and applying in their working. Also, without successfully implementing the coreset framework, it will probably not have any progress for the coreset application in the industry area. Maqbool et al. (2017) emphasized the challenge of big data in the new industrial era, which refers to a revolution towards the digital world with the quick updating of smart electric devices and mobile applications [26]. The solutions to maintain the accuracy and ef-

ficiency when dealing with astronomical data is a crucial problem nowadays. For this reason, efficiently constructing a coreset generating framework is a meaningful topic and task in many aspects. Nonetheless, coreset is relatively a novel topics which means there are not many resources or repositories on the internet that can be utilized in implementing it. In other words, the actual implementation of the coreset will start from scratch entirely and cause a considerable amount of work. Also, due to the complication of the coreset generating algorithm, writing a simple single-thread version of the coreset method in Python [45] is mainly working at the beginning. Furthermore, a more comprehensive version of coreset that supports merge & reduce computing and applying GPU [48] to conduct the parallel computing will be implemented based on C++ and CUDA [18] programming languages.

The evaluation criteria of the performance of the coreset in replacing the computing among the whole original dataset will be conducted with the classical k-means clustering algorithm. K-means algorithm is an illustrious clustering approach for all dimension of points that dramatically update the centres by calculating the centres of current clustering at every iteration through a deterministic global searching [32]. In addition, instead of using the default random initialization method, we use a more stable initialize method named k-means++ to generate better starting centres [3]. The Chapter 4 will describe the specific approaches and formula for evaluating coreset performance.

1.2 Structure of the Thesis

In total, there are nine parts in this thesis, including the abstract and introduction modules that we all already seen above. In the next chapter, which is Chapter 2, mainly content will be the background information about coreset, including the detailed introduction to the related works as well as the latest researching progress in this area. In addition, the reasons for choosing the methods and algorithms that will be applied in the implementation will be introduced in this chapter. Also, the strengths and downsides of related frameworks like CuMl and SparkML will also be discussed in Chapter 2.

In Chapter 3, I will describe the specific development and production environments, which refers to the environment used in implementing and the environment that applied in operating the program, respectively. Detailed information about all the programming languages, libraries, and development tools utilized in this coreset implementation can be found in this chapter. Furthermore, this chapter will also introduce the information about the specific cloud computing server that involved in this project. Also, an

explanation about where is our experiment data come from is also located in Chapter 3.

Furthermore, Chapter 4 will explain what methods we propose and exploit in this thesis project in a detailed format. It will mainly including two parts; the first part is about the algorithm for constructing the coreset, the second part is related to how we implement this coreset generating algorithm. What is more, the next Chapter, Chapter 5 contains the comprehensive workflows and detailed introduction related to the implementation of the coreset method. This chapter will stand on the engineer sight to describe the involved techniques during the entire implementation.

Then, Chapter 6 describes the evaluation methods that we applied in the experiments, and the detailed processing record and results can be seen in this chapter. Basically, in this chapter, we will explain what kind of experiments we did and why we select to do those kinds of experiments. Furthermore, Chapter 6 describe the experiments results with text, diagrams, and data table forms. This chapter will also discuss potential reasons for why the results occur. Also, we will compare our results with the widely-used CuML library, mainly on the running speed, cluster accuracy and performance when dealing with the situation that the data set is oversize than the GPU memory. The advantage and disadvantage of our coreset computing framework and the CuML machine learning library will be exhaustively described in this chapter.

In the end, Chapter 7 aims to discuss the shortage of our coreset method implementation and what is the future topics or works that can be done based on this project. Besides, Chapter 7 will describe what are the aspects that need to be improved in our coreset computing framework, in both theoretical and practical perspective. After the explanation of future expectation, Chapter 8 concludes the entire project, including both academic learning works, implementation works, and thesis writing works. This chapter will summarize why we set up this topic, what we have done so far, and the results that we obtained in this project.

Chapter 2

Background

Due to the more and more big data, there is a great demand for novel clustering algorithm that beyond the traditional algorithm, (1) with the ability to operating in parallel on distributed computing server or GPU computing, (2) can be deployed in the mobile device to conduct real-time computation, namely, apply online computation, (3) deal with the situation that data set size is larger than the storage of CPU or GPU memory, especially the memory limitation issue in GPU since the progress of GPU is way far behind than the innovation of CPU, in result the affordable GPU memory storage is much less than the memory storage of same level CPU.

Many pieces of research are related to proposing the new algorithm or method that can have the properties we describe above. A standard solution is to re-invent the algorithms or computational models for dealing with these novel computational scenarios, namely, developing a new algorithm or method from scratch, which means developing independently from existing solutions. In general, there are three categories of new computational model: Distributed/Parallel Computing, Streaming computing and Dynamic computing. Distributed/Parallel computing model means to minimize communication cost, and/or maximize parallelism in calculating based on multiple computing units (CPU, GPU etc.). The streaming algorithm aims to operate on a data stream using o(n) memory by processing the data chunk by chunk (or buckets by buckets) and aims to come up an approximate answer to the input data with as less usage of space as possible [42]. Dynamic computing responds to queries quickly (ideally polynomial log(n), which refers to simplifying a puzzling problem by breaking it down into simpler and smaller sub-problems in recursive or iterative manners.

Furthermore, all three computing models have some strengths related to the novel algorithm properties that we conclude above. These three computing models can be designed to utilize multiple operating servers and apply

GPU to obtain the benefits of parallelism. Also, they have the potential to be implemented to process real-time computation and ported into a lightly-computing mobile device. However, as for the third property, which is to cope with the situation when the data size is over the CPU/GPU memory, all three novel models we described above are limited. The complexity of re-designing the algorithms to suit a vast data size level is very high. And in short, these novel models are still the algorithm driven approach, which means you need to re-invent the algorithm to adopt the different data sets and problems. Comparing to above three computing models, coreset has a most powerful strength is to keep the workflows of original algorithm. Namely, the coreset does not change the operating algorithms; it only processes with the original data, which refers to summarize the big data to a small coreset and the existing algorithms or open-source libraries can be directly applicable. As a result, with less time consuming in re-inventing the existing algorithms or implement new computing models, coreset can empower the small device to handle big data.

2.1 Coreset Methods and History

Pankaj et al. (2005) proposed the term coreset to compute the smallest k balls covering a set of input points [1]. Overall, coreset is a novel data structure C that summarises the original data set (usually big data set) to a small data set. The result obtained from the coreset proved to approximate the result obtained from the whole original data set. Actually, there are different types of coreset. Composable Coresets [37] are significant classes of coreset that are especially relevant for dealing with big data. The composable property of coreset ensures that a union of a pair of coresets $C_1 \cup C_2$ is a coreset for the union of input P_1 and P_2 , and we can compute coreset C_3 for the union of $C_1 \cup C_2$ recursively [12].

2.1.1 Coreset for Clustering

Coreset is an efficient modern technique for data mining and data reduction that obtained wide popularity recently in many research fields. And there are many works on designing a coreset generation algorithm for k-means and other clustering problems, like k-median. Clustering is a classic tool in the machine learning area with various useful applications to classification, unsupervised learning, data analysis and other sub-fields. In recent years, the research related to the use of coresets to solve clustering problems has made great progress. Examples can be seen at [16, 21, 22, 25, 31].

Specifically, in [6], Vladimir Braverman et al. introduce an efficient algorithm to build a small coreset for k-median in an excluded-minor graph, and the result can easily extend to k-means clustering [6]. Furthermore, this is the first coreset construction approach that only depends on ϵ, k, and the size of excluded-minor [6]. In addition, this construction approach is also applicable in Euclidean metrics by easily applying the novel terminal embedding result proposed by Narayanan, and Nelson in [43].

We can briefly define the background problem and objective of this algorithm as follows [6]:

- First, given a weighted input points set $X \subseteq \mathbb{R}^d$ with a weight function $w : X \to \mathbb{R}_+$.
- Then, the algorithm target is searching k centers $C \subseteq \mathbb{R}^d$ to minimize the total distance cost that pair each point with its nearest center in C:
$$cost(X, C) := \sum_{x \in X} w(x) \cdot d(x, C)^2$$
- The $d(x, C) := \min_{c \in C} d(x, c)$ is the distance between point x to its closest center. And the distance function d is the **Euclidean Distance** in this thesis project.
- Then, we get a weighted sub-set $D \subseteq X$, which is an ϵ-coreset for k-means clustering on top of data set X, and we have:
$$\forall C \subseteq X, |C| = k; cost(D, C) \in (1 \pm \epsilon) \cdot cost(X, C).$$

Furthermore, constructing small coresets are interesting in clustering problems since we can solve the problem on set D instead of on original big data set X, which can lead to a vast improvement in time, space or communication complexity. Therefore, the size of coreset is one of its most significant evaluation item. Har-Peled and Mazumdar [22] firstly proposed the coreset generation method to cope with the k-median and k-means problems in Euclidean spaces. Subsequently, due to their groundbreaking research, small coresets designing and research have become an exuberant exploring direction, not only including in k-Means and other clustering approaches, but also in many other crucial computer science and machine learning problems, such as PCA problem [15], regression problem [35], density estimation [53].

In general, many coreset generation techniques are applying the idea to extend the importance sampling [29] approach and based on the excellent works of a fundamental framework proposed by Feldman and Langberg [14].

In work proposed in [6], it introduces a coreset construction approach for K-clustering problems that utilizes the concept of importance sampling method and k-means++ approach, which is the general ideas that the method we are following in this thesis project. The primary innovation of this novel algorithm introduced in this article [6] is the iterative property, which refers to firstly reduces the original n input points to approximately $O(\log n)$ reweighted points, then to $O(\log \log n)$, and go forth until the coreset size is independent of n. In each step, importance sampling is used to iteratively reduce the data size, with a crucial adaptation that reduces the number of distinct points by employing a terminal embedding. Where the terminal embedding is technically involved and relies on shortest-path separators, which is a standard tool in planar and excluded-minor graphs.

However, one limitation of using k-means++ and importance sampling approaches to construct a coreset is that this type of coreset construction methods require traversing through the whole data set for k times. Thus, despite the linear complexity with the data set size n, even for medium value size of k, constructing a coreset for a vast data set becomes very time-consuming, especially in machines that only have a small amount of computing power. Therefore, in our implementation, instead of only implementing the basic sequential computing version of coreset, we also propose the merge-and-reduce version coreset computation method that applies the parallel calculation and empower even the small GPU to cope with the computation of big data by utilizing the composable property of coreset. More details about the methods and implementation can be seen at Chapter 4 and Chapter 5.

2.1.2 Coreset Advantages

Reduction of time, memory, and communication cost through the much smaller coreset, comparing to conducting all types of processing in the original massive data set. As we mentioned, keeping the efficient and consuming-friendly in coping with the big data is a serious problem currently, and coreset is a decent way to solve both the issues of vast time and space cost in facing the big data [12].

Convert current off-line algorithms to streaming algorithms. Due to the composable property of the coreset, it is possible to utilize this fundamental property to process data piece by piece (or we can say chunk by chunk). We can easily apply the idea of merge-and-reduce and construct the coreset using a relatively small machine.

Turn existing sequential algorithms to parallel algorithms . Since we can deal with the coreset computation chunk by chunk and process the data in each chunk, it is wholly independent. We can quickly transform the

sequential computational graph to a parallel operating manner by invoking multiple threads in CPU or GPU. Stand on the implementation perspective, once the coreset computation method is successfully constructed, it is very clear to implement the merge-and-reduce version of coreset computing by following the concept of merge-and-reduce, which is to reduce the problem to two sub-problems and merge the results of these two sub-problems.

Boosting existing heuristics [12]. It is possible by running existing heuristic algorithms (that is, algorithms without provable guarantees) on a coreset, which is much smaller than the original data set. For instance, we can operate more iterations or initial values or diversely parameters on the coreset comparing to perform one iteration on the original (big) data. And the coreset is regarded as a connection between theoretical research and practical systems; see an interesting example in [11].

Optimization is the most strong advantage of using coreset when facing big data [12]. In the area of computer science, optimization is considered to be one of the most significant sub-fields, which refers to maximizing or minimizing some function relative to some data set. In addition, optimization is one of the key components of machine learning. The objective of most machine learning algorithms is to design an optimization computing graph and learn the parameters in the target function from the given (training) data [58]. For instance, in most cases of deep learning, the goal of training is to search for the best parameters by computing the difference between output and fixed target to lead the model to learn the data features. Usually, optimization is very time-consuming for many problems. For instance, when the clustering size k is part of the input, the k-means clustering problem becomes NP-hard [38]. Whereas, solving the optimization tasks or its approximation problem on the coreset proved to yield a solution that approximate to the solution that calculated from the original (big) data set, or sometimes after appropriate post-processing [12]. In k-means clustering problem, the near-linear size in (k/ϵ^2) composable coresets could be applied to generate $1 \pm \epsilon$ approximate solution in a time complexity of $O(ndk)$ [15]. In this thesis project, we also evaluate the performance of the coreset by the classical k-means clustering problem.

2.1.3 Coreset Types

In general, most of the coresets are designed to obtain a multiplicative $(1+\epsilon)$ approximate solution for the desired objective value of every query. We can conclude coresets into two types, strong coreset and weak coreset.

Strong Coreset [42]: Define A be a subset of universal set U, define C be the potential solutions set. Let $f: U \times C \to \mathbb{R}^{\geq 0}$ be a non-negative distance

measurement function. We have an $\epsilon - coreset$ (Strong coreset) $S \subset U$, if for all potential solutions $c \in C$, it have

$$|f(A, c) - f(S, c)| \leq \epsilon \cdot f(A, c)$$

Overall, Strong Coreset is able to approximate any input queries in the query set. Therefore, Strong coreset is composable and approximate solution to the optimal query of the original input data is usually accessible to be constructed in a strong coreset [12]. Look more information and examples in [1, 9]

Weak Coreset [42]: Weak Coreset is another category of coresets. However, there are many divergent definitions of weak coreset, which leads weak coreset is not as straightforward as strong coresets. In brief, weak coreset S only prove that a solution computed on top of S is a $(1 + \epsilon)$ approximate solution to the optimal solution generated in the original data set A. As we mentioned, be aware that there is no unifying agreement on the accuracy guarantee of weak coresets, and many conflicts occur in different articles. To conclude, instead of approximating all the possible queries, Weak Coreset C is limited to resemble only a portion in all queries. [12]

2.1.4 Coreset Construction Methods

We mainly introduce four types of coreset construction methods in a general form based on the survey proposed by Feldman et al. (2020) [12]. Note there are still many other coreset construction methods that not mentioned in this thesis. More information about different types of coreset generation methods can be found in [7, 33, 42].

Uniform Sample. Uniform sampling on top of the input data is apparently the most typical coreset. Basically, just generate a small subset from the original data set and ensure that every point has the same probability of being picked out. Uniform sample is naturally a very decent competitor with the coreset that generate using other methods. Compared to other construction methods, although Uniform Sampling spends sub-linear time constructing a coreset, it leads to loss of more minor but essential points, usually called distant clusters. Besides, uniform sampling does not offer $(1 \pm \epsilon)$ multiplicative error proof like other types of coresets construction method. To conclude, Uniform sample is a fast and straightforward way to create a 'naive' coreset, a suitable baseline for exploring different types of coreset construction.

Importance Sampling decreases the $\epsilon \cdot n \cdot cost(P, x)$ additive error in uniform sampling to $\epsilon \cdot cost(P, x)$ by replacing the sampling weights. In more detail, this means that instead of using the same weights on top of all the

input points, re-weighting each point by its sensitivity that usually calculated by its distance to the center points. In statistics, importance sampling is a common technique for estimating properties of a particular distribution. In this thesis project, the coreset generate method that we implemented is basically based on the importance sampling concepts, which can be seen in detail at Chapter 4.

2.2 K-means Clustering Problem

K-means clustering is an approach for vector data classification. K-means was originally created from the signal processing area, aiming to separation n input points into k different clusters. Specifically, each point classifies to the cluster with the nearest average value (cluster centres or cluster centroid). The k-means problem [32] is computationally complex, an NP-hard problem; however, efficient heuristic algorithms can converge quickly to an optimal local result [27]. K-means is an unsupervised algorithm with less connection to the k-nearest neighbour algorithm, a famous supervised machine learning approach for data classification problems that are usually confused with k-means since a similar name.

In definition, input a collection of points $(x_1, x_2, \ldots, x_n)$, where each point is a real vector with d-dimensional shape, k-means clustering algorithm designed to divide the n points into k ($\leq n$) clusters, $S = \{S_1, S_2, \ldots, S_k\}$, the objective function of k-means method is to minimize the sum of square distance between points and its closest cluster center:

$$\arg\min_{S} \sum_{i=1}^{k} \sum_{x \in S_i} \|x - \mu_i\|^2$$

2.2.1 K-means Method

The terminology 'k-means' was first introduced by James MacQueen in 1967 [36] aims to classification problem and analysis of Multivariate Observations. The standard algorithm of k-means was designed firstly by the researcher Stuart Lloyd during his work in the well-known Bell Labs in 1957 as an approach for modulating pulse-code. However, it was not published as an available article for other researchers until 1982 [32]. We usually regard the standard k-means algorithm as a 'naive' version of this algorithm since there exists better variant of k-means algorithm which uses iterative refinement as a improvement [51].

Given an initialization centers of k-means $m_1^{(1)}, m_2^{(2)}, ..., m_k^{(1)}$ (see below), this algorithm conducts computing by iterating following steps: [32]

Step A: Assign each point to the cluster with the shortest distance between this point and the cluster center. Furthermore, the distance was measured with squared Euclidean distance function:

$$S_i^{(t)} = \{x_p \colon \left\| x_p - m_i^{(t)} \right\|^2 \leq \left\| x_p - m_j^{(t)} \right\|^2 \forall j, 1 \leq j \leq k\},$$

Note that each x_p is assigned to exactly one $S^{(t)}$.

Step B: Recompute the centers (means) for the points that assigned in every cluster.

$$m_i^{(t+1)} = \frac{1}{|S_i^t|} \sum_{x_j \in S_i^{(t)}} x_j$$

Usually, we consider that the k-means algorithm reaches convergence when the assignments no longer change or have a very tiny altering. Note that k-means use the (squared) Euclidean distance function to calculate the distance between the points; change to other distance function may lead to a long convergence or even prevent the algorithm from converging.

2.2.2 K-means++ Initialization

In the standard k-means method, the centers was initialized randomly, which means they are picking out from the input points uniformly. In practice, sometimes poor clustering will be found by naive random initialization technique. In 2007, David Arthur [3] proposed k-means++ algorithm for better selecting the initialisation centers of k-means algorithm. See more details about k-means++ algorithm at Section **4.2**.

2.2.3 K-means Application and Implementation

The k-means and k-means++ approaches have been applied in many problems and applications. Lee et al. [30] introduce an interesting usage of applying k-means++ to generate a topographical cluster of pictures by their numeric values of the latitude and longitude. Oyelade et al. [49] report a novel application of predicting students academic performance (GPA) by utilizing the k-means clustering.

There are also various software and open-source implementation of k-means and k-means++ methods that proposed by many researchers and engineers. Some representative examples include:

Matlab has a embedded k-means approach that default use k-means++ initialization method.

OpenCV supports k-means for processing pictures pixel values.

R language has k-means method and support k-means++ by 'flexclust' package.

Scikit-learn includes an efficient k-means implementation in single thread that applies k-means++ initialization by default.

CuML contains k-means implementation in multiple threads setting with GPU computing; it shares a similar API with Scikit-learn.

2.3 Machine Learning Libraries and Frameworks

Machine learning is a combination of methods in data analysing that able to automatically create computing models. Machine learning is regarded as a sub-area of artificial intelligence, and the basics idea of machine learning methods is to let the model automatically extract knowledge from data. Namely, empower the machines to recognize different patterns by themselves and come up with decisions without strongly influence of human intervention. Usually, machine learning algorithms, including both supervised and unsupervised methods, are complicated to implement efficiently. Without a framework, machine learning is a nightmare. It is not wise to build everything from scratch, especially when you are in a business environment. Recently, many brilliant machine learning frameworks have been created and proposed to solve different kinds of problems in machine learning scenarios, including speech recognition, computer vision, data classification and data dimension reduction.

Scikit-Learn is a well-used python third party library adopted by many researchers and engineers to solve machine learning problems in many different scenes. It handles both supervised and unsupervised learning and has comprehensive and detailed documentation for every potential doubt. It is a very successful and efficient machine learning framework on the CPU side. However, a single thread calculating mechanism is a significant limitation, leading to a relatively slow operating speed when encountering astronomical data. See more information about Scikit-learn in [50]

CuML is a collection of efficient, user-friendly machine learning algorithms with GPU acceleration that designed for dealing with machine learning and data science problems. The API of CuML is similar to the API in Scikit-Learn, which means users can quickly transform the code used to train the models in Scikit-Learn to the code in invoking the methods of CuML. CuML enables software engineers, data scientists and researchers to fast conduct traditional tabular machine learning tasks on GPUs without learning deeply of CUDA programming knowledge. For vast data sets, those GPU-based implementations in CuML can operating 10-50x times faster than their CPU equivalents. This article [44] reports the acceleration of CuMl in processing the machine learning problems. Furthermore, CuMl supports various categories of machine learning algorithms, including but not limited to clustering, dimensionality reduction, regression models and classification algorithms.

Spark is an integrated analysis engine proposed by Apache for processing and analysing large-scale data set. It supports high-level APIs in R, Python, Java, and Scala. Furthermore, Spark a supplies a suite of advanced tools in MLlib for coping with machine learning problems. Spark aims to simplify and standardize the solution pipeline of practical machine learning problems and provides classification, regression, clustering, feature extraction, dimensionality reduction, and tuning ML Pipelines. Spark has sorts of well implemented approaches in clustering. The spark.mllib implementation includes a parallelized variant of the k-means++ named Scalable k-Means++ that proposed by Bahmani et al. (2012) [38]. This function will be the most important performance benchmark for my coreset-based implementation in thesis research, more information of this function can be found at `https://spark.apache.org/docs/latest/mllib-clustering.html`.

Chapter 3

Environment

Since we are mainly focused on the implementation works of the coreset method, including the implementation in both CPU and GPU sides, there are many different operating environments through the whole processing of coreset generating. Mainly contains the Linux and Mac OS development environment, Python running environment, CuML environment, C++ and CUDA programming and running environment and GPU environment. In addition, there are many other libraries or tools that have been applied in the implementation - for example, utilizing Matplotlib to automatically create the diagrams, using Trust to implement the high-performance methods with minimal programming effort, applying Pandas and Numpy libraries in dealing with the computation of the array in the Python side. In addition, the version control system Git is also used in this project to simplify the workflows.

3.1 Development Environments

Throughout the developing process of this coreset construction method, we mainly write the program of our project in the Linux operating system and Mac OS system. They are both based on the Unix system and shared many commands in the terminal code, which leads to high compatibility on these two systems.

Linux is an open-source operating systems for modern computers based on the Unix operating system. Linus Torvalds firstly proposed this operating system and released first version of Linux kernel on September 17, 1991. Actually, Linus is Finnish, and he started to write this fantastic open-source project when he was a student at the University of Helsinki. This is actually one reason that I decided to cross the continents and grasped my master studying at Aalto University, Finland.

In more detailed form, the code was written in the cloud server of Aalto University, which is: **Linux version 5.4.0-72-generic (buildd@lcy01-amd64-019) (gcc version 9.3.0 (Ubuntu 9.3.0-17ubuntu1 20.04))**. Furthermore, I used Vim as the text editor. Vim is convenient to be used either from the command line interface or as an independent application in the graphical windows. Vim is also an open-source software that free to download and use, and is distributed under a license that includes some charitable software terms to encourage users who are fond of this software to donate for the children in Uganda. The specific version of Vim that used in this project is **VIM - Vi IMproved 8.1 (2018 May 18, compiled Apr 15, 2020, 06:40:31)**.

MacOS is a well-applied operating system with graphical windows that developed and marketed by Apple Inc. since 2001. It is the primary and default operating system for Mac computers of Apple. Since the popularity of Mac laptop, there exists a very strong community in MacOS, which supply all kinds of support of software and interface. The specific computer that I mainly used in the development of this project is below:

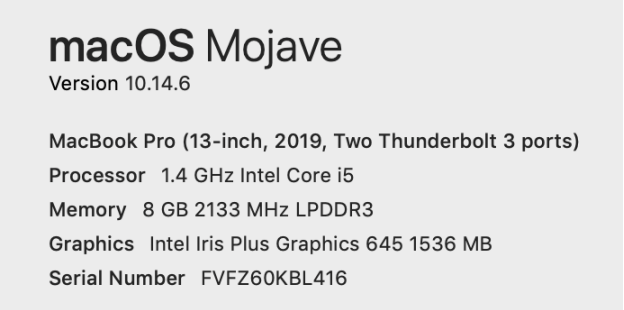

Figure 3.1: The detailed info about the Mac computer used in this thesis project

Furthermore, I utilized PyCharm IDE and Visual Studio Code IDE in developing the python and C++/CUDA versions of coreset, respectively. PyCharm is an commonly used integrated development environment (IDE) in the computer programming area, mainly for writing the Python language. It is created by the Czech company JetBrains (usually called IntelliJ).Visual Studio Code is a free software source code and text editor developed by Microsoft for Windows, Linux and MacOS. Common characters include support for debugging, smart code completion, syntax highlighting, code snippets, code refactoring, and embedded support for Git. In addition, users can change themes, keyboard shortcuts, preferences, and install extensions for

additional functions.

In addition, Git and GitHub have been used throughout the whole development procedure. A typical workflow is implementing the code at the MacOS environment first and using Git to store and push the commit; after successfully backup the log information, we pushed it to the remote repository. Afterwards, we pull back the latest code in the Linux environment by git pull command. Then, we can efficiently conduct the experiments and testing with the latest code in the Aalto Linux server, which is much over perform the computing performance in my personal laptop.

3.2 Python Environment

Python is a programming language designed by Guido van Rossum as an interpreted high-level for general-purpose programming tasks. The design philosophy of Python applying the notable use of significant indentation to emphasizes and improve code readability. Python aims to provide programmers a clear programming logical for dealing with small and large-scale projects with the support of its explicit language grammar as well as its object-oriented property. [28].

Besides, one of the most significant reasons why Python is well-used in many different fields is its strong and active community. The detailed and comprehensive documentation and information about the standard Python functions, but there are also are countless fantasy third-party libraries that aim to help programmer efficiently implement the application in all kinds of areas. Many famous third parties, including Flask/Django, build the website; Numpy/Pandas in the data science area; PyTorch/Tensorflow in building and training the deep learning neural networks models. In general, it is widespread to search for some well-implemented libraries that can handle the same problem before starting to write the implementation to the problem from scratch.

As for the specific Python version used in this thesis project, in both Linux and MacOS development environment, we used **Python 3.8.5**, which is one of the latest but robust python version. The primary third-party libraries that been used in this project including:

- CuML (0.17.0)
- Numpy (1.19.5)
- Pandas (1.1.5)

- Scikit-learn (0.24.1)
- Scipy (1.6.0)
- Cudf (0.17.0)
- Cupy (8.0.0)
- Numba (0.52.0)

In total, there are roughly 60+ different third-party libraries in the Python environment of this project.

3.3 C++/CUDA Environment

C++ is created by Bjarne Stroustrup as a general-purpose programming language for dealing with different programming tasks. The grammar and functions of C++ are based on the C programming language and usually regarded C++ as "C with Classes". With the passage of time, C++ programming language has received many significant extensions, and modern C++ now has object-oriented, universal and functional functions in addition to functions for low-level memory operations. The C++ language has two key components: direct mapping of hardware elements mainly provided by a subset of the C programming language, and zero-overhead abstraction based on these mappings. Providing hardware access and abstraction at the same time is the advantage of C++, and executing it efficiently makes it different from other programming languages [57].

The C++ programming language was first standardized as ISO/IEC 1488:1998 in 1998, and then improved by the C++03, C++11, C++14 and C++17 standards. The latest C++ 20 standard replaces the new features and standard library of these versions. It is almost always implemented as a compiled language, which means that when running C++ code, we first need to compile the code to process machine instructions. Many manufacturers provide C++ compilers, including the Free Software Foundation, IBM, Microsoft, Intel, and Oracle, so it can be used on many platforms. [56]. In this project, we applied **gcc/8.4.0** compiler in the Linux system to compile the C++ code.

CUDA (Compute Unified Device Architecture) is a parallel computing platform and application programming interface (API) model, which was founded by NVIDIA, a multinational technology company headquartered in Santa Clara, California. CUDA allows programmers and engineers to implement programs running on GPU using graphics processing unit (GPU)

supporting CUDA[18]. In addition, the CUDA platform is designed to be used with other CPU-side programming languages such as C, C++ and Fortran. Compared with other GPU programming languages, this accessibility and compatibility is just the process of parallel programming experts using GPU resources. CUDA enables software developers to use the power of GPUs to accelerate the parallel parts of computationally intensive applications. In this thesis project, we use the CUDA of version **cuda/11.0.2**.

3.4 GPU Environment

GPU (graphics processing unit) is a particular electronic circuit designed to manipulate and alter memory to accelerate computation rapidly. Recently, there is a huge progressed demand for GPU based program, since it can simply speed up over 100+ times than the identical CPU program in dealing with the big data situation, e.g. in machine learning/deep learning assignments, which is why GPU has become an indispensable device in the research community of computer science and many other areas. We mainly utilized the Aalto high-performance computing cluster Triton which is currently coordinated from within the School of Science of Aalto but serves all researchers and students (require an application) of Aalto. On Triton, there are a large amount of NVIDIA GPU cards from different generations. Triton GPUs are not the typical desktop GPUs in personal computers but specialized research-scale server GPUs with colossal memory, high bandwidth and technical instructions. For scientific purposes, GPUs on Triton generally outperform the best desktop GPUs. Figure 3.2 describes the GPU cards that available on Triton. Find more information about Triton cluster and Aalto scientific computing here: `https://scicomp.aalto.fi/`

We mainly request the Tesla P100 GPU card with Pascal architecture and 16 GB memory in our experiments.

3.5 Environment of Other Tools that Used

As a comprehensive thesis project, this thesis project includes both implementation and thesis writing works. Therefore, besides helpful programming tools, many other tools in dealing with text writing, code version control, and visual meeting are also used during this project.

We mainly use LaTeX language in writing the thesis and many other documents related to this thesis project. LaTeX is a famous software system for friendly and neatly document preparation, which is widely used in academia

Card	total amount	nodes	architecture	compute threads per GPU	memory per card	CUDA compute capability
Tesla K80*	12	gpu[20-22]	Kepler	2x2496	2x12GB	3.7
Tesla P100	20	gpu[23-27]	Pascal	3854	16GB	6.0
Tesla V100	40	gpu[1-10]	Volta	5120	32GB	7.0
Tesla V100	40	gpu[28-37]	Volta	5120	32GB	7.0
Tesla V100	16	dgx[01-02]	Volta	5120	16GB	7.0

Figure 3.2: Available GPUs and architectures on Triton

for the communication and publication of scientific documents in many fields [19]. Many different editor applications for writing a paper in LaTeX form, including TeXmaker, Kite, TeXStudio, Emacs, LaTeX Workshop, Overleaf. There are available LaTeX editors in all common-used operating system. We use **Overleaf** as the editor tool in thesis writing due to its convenience and collaboration. Overleaf is a collaborative cloud-based LaTeX editor used for writing, editing and share academic documents.

We applied **Git** and **GitHub** throughout the project to easily control the code on different platforms and keep the program up to date. Git is a free and open source distributed version control system that can accurately and effectively handle online projects from small to large. GitHub is an Internet hosting provider that uses git for software development and version control. It provides distributed version control and source code management functions for Git.

Moreover, due to the keeping serious situation of the COVID-19 pandemic [52], this six-month thesis project was completed entirely online. Therefore, under the epidemic prevention requirements of Aalto University and the Finland Government, online meeting replace the physical arrangement to reduce unnecessary contact. We mainly use the online meeting software **Zoom** for weekly progresses discussion meeting. Zoom is the leading software in the field of modern video communication. It has a simple and reliable video and audio conference cloud platform.

Chapter 4

Methods

In brief, We make use of the importance sampling method that was proposed in [14], which was further improved in [15] and [6]. Essentially, [14], and [6] simplified and generalized the importance sampling method in [14], so that the technique may be applied on various types of metric spaces and that the coreset size is slightly improved in certain parameter. We will follow the coreset construction algorithm for the k-means problem proposed in [6] in this thesis project.

4.1 Importance Sampling

Before we go deep into the details of the construction of the coreset, we introduce the importance sampling method used [6] in the construction algorithm. In contrast to uniform sampling, which every point in the data set has the equal probabilities of being drawn; the importance sampling draws samples based on the likelihood of every sample. In this case, uniform sampling can be regarded as a particular type of importance sampling that every point has the same weight, usually initialized to $\frac{1}{|N|}$. At an abstract level, the importance sampling method used in [6] contains two steps:

- 1. **Computing probabilities:** for each point x in the points X, compute p_x, with $\sum_{x \in X} p_x = 1$ and $\forall x, p_x \geq 0$.
- 2. **Sampling:** draw N (samples size) independent sample x from X, each x with probability in p_x.

4.2 K-means++ Algorithm

K-means++ algorithm is also used in this coreset construction method to obtain the k approximate centers at the beginning. The workflows of the k-means++ initialization algorithm can be described as follow:

Let $D(x)$ denotes the shortest distance from a data point to the closest center that already been chosen. Then, the k-means++ algorithm is defining as follows [3]:

a. Choose the first center c_1 uniformly from the input points X.
b. Pick a new center c_i, from point $x \in X$ with probability:

$$\frac{D(x)^2}{\sum_{x \in X} D(x)^2}$$

c. Repeat step b, until we obtained all of the k initial centers

4.3 Coreset Construction Algorithm

In this section, we describe the workflows of the algorithm that we used in constructing the coreset. Overall it is based on the k-means++ initialization method and importance sampling technique. The specific algorithm is described below:

Coresets for K-means Clustering:

- Utilize k-means++ method to compute an $O(1), O(1)$-approximate solution C^{apx} for k-means clustering on original data set X. Namely, use K-means++ initialization technique to find the first k centers of X.

- for each $x \in X$, let

$$\sigma_x := w_X(x) \cdot \left(\frac{(d(x, C^{apx}))^2}{cost(X, C^{apx})} + \frac{1}{w_X(C^{apx}(x))}\right)$$

  Where w_X is the weights of $x \in X$, $w_X(C^{apx}(x))$ refers to the amount of points in the cluster that x belongs to. Specifically, in this thesis project, d is point to Euclidean distance function, which means that we use squared Euclidean function to measure the distance between the points in X.

- then we compute the probabilities of each point based on the value of σ, for each $x \in X$, let

$$p_x := \frac{\sigma_x}{\sum_{y \in X} \sigma_y}$$

- after getting the probabilities, draw N (sample size/coreset size) independent samples from X, follow the probability distribution of $(p_x : x \in X)$. Here the coreset size N is an important parameter; usually, in the experiments we set N manually, for instance, we can test coreset size between $[100, 100000]$.
- let D notates the set of the samples, and for each $x \in X$, set a weight:

$$w_D(x) := \frac{w_X(x)}{p_x N}$$

- now, the weighted set D is the coreset, return D.

4.4 Merge & Reduce

The coreset we generate using the method mentioned above has the significant composable property, which means **(coreset**$(X) \cup$ **coreset**(Y)**) is a coreset for** $X \cup Y$.

Therefore, we can simply apply the merge-and-reduce technique to recursively compute the coreset. This will bring us the strength for parallel computing the coreset of different parts of data and then combine them, especially when the original set is super massive and cannot fit in the GPU/CPU memory at one time.

The general idea is first to split the big initial data set into smaller data chunks (or we can call it buckets) and compute the coreset of the combination of two chunks to get an upper-level chunk, repeat this until we have only one chunk at the professing queue. Finally, we compute the coreset upon this chunk to obtain the final coreset.

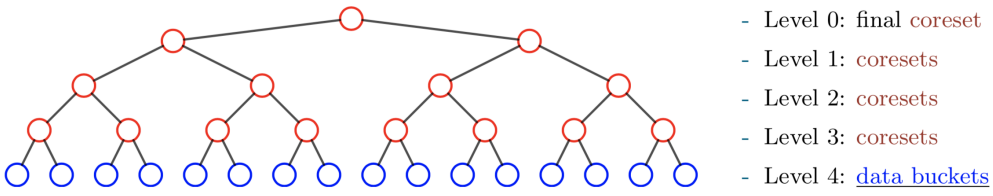

Figure 4.1: Computation graph of merge-and-reduce version coreset method

Chapter 5

Implementation

This thesis project efficiently implements the whole pipelines for the coreset construction method, including data loading, data preprocessing, k-means++ method, coreset construction, and evaluation programs. We involved the implementation procedures with Linux operating system and MacOS, using different development tools like Vim, Visual Studio Code, PyCharm, etc. We use GitHub as the cloud backup for our project and applying Git to record project progress in real-time. Furthermore, our implementation is cross-language that utilize Python, C++, and Cuda. Precisely, we build a naive Python single-thread version coreset computation method to prove that our method works; then we use C++ and Cuda to implement the multi-threads parallel version coreset method; and finally we use CUDA to implement the merge-and-reduce version of coreset construction method. Overall, our project is a cross-environment implementation, and we utilize the latest development tools and platforms in working on this project. Also, we conduct the benchmark against the well-known CuML library.

Our principal contribution is effectively implemented a merge-and-reduce version coreset computation method which aims to cope with the situation the input data is far more oversize than the GPU memory. In addition, we fill the blank that the modern well-known machine learning libraries like CuML and Spark do not use the coreset methods in their solution of machine learning problems.

5.1 Data Loader and Preprocess

In general, the data for clustering is in the CSV or other similar text format that can easily convert into CSV format. Therefore, we define the data I/O interface to cope with the reading/writing for CSV files. The C++ code of

data loader and preprocessing is located in the **dataloader.cpp** file in our repository. Specifically, we use C++ standard **vector** data structure as the container to store the data points, and we use the file reading interface that supported in $\langle fstream \rangle$ head file to read/write CSV files.

5.1.1 Read/Write CSV Files

Read CSV files into C++ vector is the first step in the entire procedure of coreset construction. We implement the CSV file reading function by using the files I/O interface in $\langle fstream \rangle$, which is embedded in C++ language. Basically, the CSV file loading program read data line by line; each line is a point with multiple different columns, then split the line by the separation symbol ',', finally the program convert the string type values to numerical type values. Noted that we use the C++ template to implement the data loader function, which means that we can flexibly choose different numerical types when creating the object of the input data vector, for instance, float, double, int types.

Instead of only read the CSV format data into 2-dimension form C++ vector, there is also a function to read the CSV data as 1-dimension form. This is instrumental when it is necessary to keep the input points vector in a 1-dimension structure, which is common in the CUDA program since CUDA based GPU computation can only accept 1-dimension arrays.

Also, there are 1-dimension and 2-dimension forms CSV writing function, which is designed to record the coreset output back to CSV format. This is because coreset is a weighted points, and it is convenient to be stored in CSV format.

5.1.2 Data Normalization

Usually, the distribution of the clustering data is vast and depends on the domain of the dataset. In some situations, for example, in dealing with open street map data [41], the columns including the longitude and latitude of a location, which is common in the range of [-180, +180] for longitude and [-85, +85] for latitude. And in case we have a considerable amount of points in some open street map files, and we need to sum the sensitive values to compute the probability of every point in the computation of coreset, it will lead to a risky of numeric overflow in C++ or Cuda. Therefore, it is necessary to apply a normalization before we conduct the computation of the coreset.

The normalization program in our project convert every value in the points to range [0, 1] based on its original value and other points value in

the axis of the same column:

$$Norm(x_j^i) = \frac{x_j^i - \min_{y \in X} y_j}{\max_{y \in X} y_j - \min_{y \in X} y_j}$$

where i $(0 \leq i \leq |X|)$ is the index of x in X, and j $(0 \leq j \leq dimension)$ is the index of column in every point.

5.2 K-means++ Implementation

Based on the method that described in Section 4.2, we implement k-means++ initialization method in both C++ and Cuda programming language. The C++ version is a CPU side version or single-thread version that does not utilize GPU's parallel computing strength. Furthermore, after we correctly complete and test the basic single-thread version k-means++, we follow the same workflow and port the CPU version to Cuda based GPU version k-means++ initialization method.

In more detail, when computing the distance between every x in X to its closet center C, we let each GPU thread conduct the computation for one point in X. Since the calculation to each x in X is entirely independent, we can utilize many threads to compute them simultaneously to benefit from parallel.

In each iteration, we first use multiple threads to efficiently obtain the probability of each point to be selected as a center. Afterwords, we apply the random function embedded in the C++ $\langle random \rangle$ header file to randomly take the new center until we get k centers. Furthermore, we use the squared Euclidean distance function to measure the distance between two points.

The input to the k-means++ initialization function contains points, the weights of points and the dimension of points; the output is the centers array. And we set the block size to **256** in the GPU computation.

5.3 Coreset Implementation

As we mentioned before, coreset is a weighted points set, which means it contains the information in the values and matters about its weights. In order to easily process with weighted points, we define a **Point** class in the **points.cpp** file, which is a data structure to store the weighted points. It contains many useful methods, for instance, **AddPoints, SetWeights, GetValues** and so on. We use Point class to store the weighted points in 2-dimension form. Also, we offer a **FlatPoint** class to support representing

weighted points in 1-dimension form, which is necessary for the merge-and-reduce version coreset method implementation.

The coreset construction method in our project is implemented in CUDA language. Our method benefits from parallelism by applying the native CUDA kernel methods and the **Thrust** parallel algorithms library. We use kernel functions to compute the σ, **probabilities** and **weights**, and we apply the **reduce** algorithm in Thrust to compute the sum of σ and doing other summarizing calculation.

In general, the workflows of coreset computing methods starting from invoking the k-means++ initialization method to obtain k centers. Then, using the k centers to compute the σ for each point. Afterwards, based on the values of σ, we calculate the probabilities of every point. Finally, we use the random function that implements in CUDA by ourselves to sample the points by the probabilities distribution. This random sample function has mainly applied the ideas of prefix-sum and random distribution. Basically, we conduct the prefix-sum to get the range of every point, and we draw a random number in the range of $[0, \sum_{x \in X} p(x)]$, and then pick the point that corresponding range contains the value of the random number.

5.4 Merge & Reduce Implementation

The Merge-and-Reduce version of the coreset construction method is the main contribution in this project. We implement it in CUDA language, basically follow the algorithm that we described in Section 4.4. We apply the recursive idea in implementing the merge-and-reduce. In brief, the program will firstly split the whole dataset into many much smaller data chunks. And it will combine two data chunks and conduct the coreset computation upon the union of two chunks to get a coreset. Then we regard the coreset that get in the previous step as a 'chunk', repeat the coreset computation until we only have one 'chunk', then we get the final coreset by calculating upon the root chunk.

In the implementation of merge-and-reduce, we invoke the coreset construction function that defined in the **coreset.cu** file. Instead of using the construction method that output a 2-dimension weighted coreset, the merge-and-reduce program invokes the coreset method that outcome the 1-dimension weighted coreset to reducing the time-consuming in transforming the dimension of the array. In more detail, since the CUDA can only receive the one-dimension arrays as the input, we need to flat the arrays before we port them into GPU; therefore, we can save the time in merge-and-reduce process by replace two-dimension coreset output to one-dimension output

form.

Furthermore, when talking about 'split' data, we are not really split the data into smaller pieces. Instead, we split the indexes range and acquire the related data chunk by its specific indexes range. Using this strategy prevent our program from a bunch of time-consuming works of vector reproduction and also save many memory space in running.

Chapter 6

Experiments and Results

This chapter contains detailed information about what kind of evaluation we did for the coreset and the corresponding results. In short, we test the coreset accuracy versus coreset size, the coreset generation speed with the evolution of original data size, and the merge-and-reduce version coreset implementation with the big data that oversize the GPU memory.

6.1 Evaluation Data and Methods

6.1.1 Data Sets

The experiments was conducted in three different data sets with the different domains, dimension and size.

- **Heterogeneity Activity Recognition Data Set** [55], this is an open-source data set that available in the UC Irvine machine learning repository. This is a data set from the smart electrical devices like smart photos and smartwatch that aims to record the statistics of humans' activities in the real-world context. There are four similar files in this data set, and we select the Watch_gyroscope.csv file in our experiment. In more detail, this file includes 3205431 points with five attributes in each point, which refers to have the dimension of **(3205431, 5)**.
- **US Census Data (1990) Data Set** [39], this is a high dimensional data for classification and clustering problem. In total, it has 68 different numeric attributes, and it has 2458285 points. Therefore the dimension of this data set is **(2458285, 68)**. The principal reason for

choosing this data set is to utilize its high dimension property to test the coreset performance in the high dimensional data set.

- **Open Street Map** [46], which is a public repository that aims to generate and distribute accessible geographic data for the whole world. Basically, it supplies detailed position information, including the longitude and latitude of the places around the world. We mainly focus on the geographic data of European countries in the experiments of this project. More specificly, we download and extract the osm location data for nine countries, including Finland, Sweden, Denmark, Norway, France, Germany, Netherlands, British, Poland. Then we combined this nine osm file to a single big CSV file that is in the shape of **(1156442555, 2)**. For convenience, we named this big CSV file as all-latest.csv. Furthermore, we conduct the coreset generation speed versus data size among this file by sampling increasing size of sub-set in this big data.

  Furthermore, we also grasp the osm data for Hong Kong, China, which has the dimension for **(2366214, 2)**. We conduct the coreset accuracy in this data set and the first two data sets that we described above.

6.1.2 Evaluation methods

The evaluation for the coreset construction method contains two parts: 1. test the coreset accuracy in k-means clustering problem by comparing the results of k-means objective values with CuML, 2. test the coreset construction speed with different size of input data.

- In the accuracy evaluation, first, we use our coreset method to get the small coreset of the big original data. Then we utilize the k-means function provided in CuML on top of coreset to get k clustering centers. On the other hand, we also get k clustering centers among the original big data by the same method k-means in CuML. Therefore, we have two sets of k centers that one of which is the clustering result in coreset, and another one is the result in the original data set. Finally, we compute the sum of objective values in the original data with both two sets of centers separately. Set the two objective values sums are obj_c (objective values for coreset) and obj_o (objective values for original data set), then we define the relative error of the coreset method:

$$error = \frac{obj_c - obj_o}{obj_o}$$

Note that we do not use the absolute value here, which means the error can be negative. And suppose the error is a negative value; in this case, it means the clustering result generated upon the coreset is even better than the result from the original big data.

We mainly conduct the experiments in the same data set with different coreset sizes to test the influence of coreset size on its accuracy. The values of coreset size that we used in this experiment are: **100, 200, 500, 800, 1000, 2000, 5000, 8000, 10000, 20000, 50000, 80000, 100000**.

- We test the coreset generation speed along with the change of input data size. This experiment is conducted in all-latest.csv file, which has more than one billion points. Furthermore, the sampling strategy is used here to get the different size of sub-sets in the same file. We select the test input data size to:
$\mathbf{10^4, 5 \cdot 10^4, 10^5, 5 \cdot 10^5, 10^6, 5 \cdot 10^6, 10^7, 5 \cdot 10^7, 10^8, 5 \cdot 10^8, 10^9}$.

6.2 Coreset Accuracy

For the computation of the relative error of the coreset, we repeat the same experiments five times in total, and we compute the average relative error of the coreset as the final error. We test the coreset accuracy with three different data set with relatively low, medium and high dimension, which refers to the dimension of 2, 5 and 68, respectively. Note that we set the clustering size to 5 in our experiments. We plot the results as a diagram, see it below:

There are several impressive results that can be seen on the diagram.

- a. Even with a small coreset size $N = 100$, our coreset method reach a tiny error that below 5 per cent in both three files. This indicates that our corset construction method can lead to a very accurate result in both low/medium/high dimension clustering data for the k-means problem.

- b. In a general trend, when increasing the coreset size, the relative error will also decrease, which lead to better accuracy of the coreset. Particularly, along with the range $[100, 8000]$, there are apparent trends of error decreasing when the coreset size increases in all three data sets. Therefore, we can claim that this phenomenon indicates that the coreset accuracy in practice benefits from the growth of coreset size when the size is less than 8000.

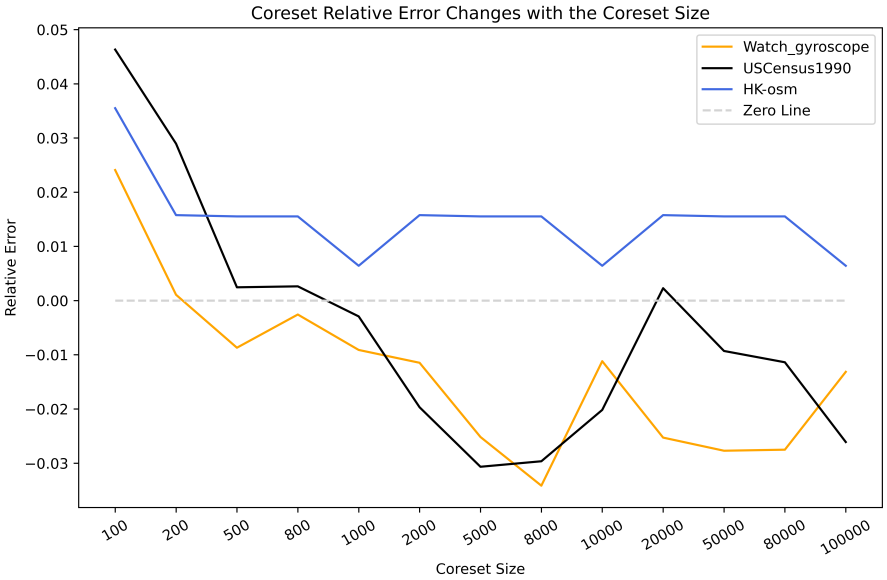

Figure 6.1: Coreset Relative Error Changes with the Coreset Size in data set Watch_gyroscope, USCensus1990, and hk_osm

.

- c. Furthermore, we can see that when the coreset size is upper than 8000, and it seems increasing the coreset size will not raise the accuracy of our coreset method. Especially, the relative errors of Watch_gyroscope and USCensus1990 rebound to a higher error when the coreset size keeps growing to a higher level. This is an interesting phenomenon; We attribute this phenomenon to the randomness in the construction of coreset. Since the error values are all at a relatively low level in our experimental results, the slight rebound caused by randomness is acceptable.

- d. The most exciting result in this experiment is: in the condition that coreset size range between $[800, 20000]$, the results of Watch_gyroscope and USCensus1990 both lead to a negative error, which means our coreset method outperforms the CuML in terms of clustering accuracy. This is an awe-inspiring result; it indicates the possibility of obtaining even a better result with less time/space-consuming than the result computed from the original big data set.

Overall, our experiments show that the coreset method generates a compelling result in the k-means clustering problem. We compare our results

with the results computed by well-used CuML libraries. It even shows that our coreset method can outperform this mature machine learning framework in some situations. As a result, we undeniably claim that the coreset method can obtain accurate results for the k-means clustering problem in practical environments.

6.3 Coreset Construction Speed

We test our merge-and-reduce implementation of coreset method performing speed with the input data size raising from 10000 to 1000000000. The coreset size is fixed to 8000 since we find that the coreset size to 8000 will usually produce the nearly best result in the previous experiments. In addition, we set the clustering size to 5 just as in previous experiments. And we conduct this experiment by randomly samples different size of sub-sets in the combination of osm files of nine European countries, which is the most extensive file in our experiments with **1156442555** points. The diagram of the coreset construction speed along with the growth of input data size is located below:

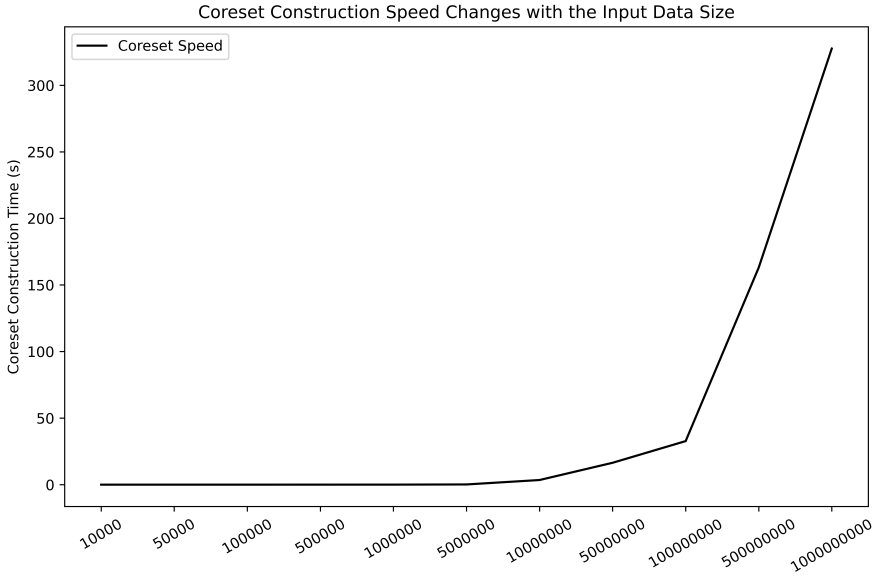

Figure 6.2: Coreset Construction Speed along with the input data ranges from 10^4 to 10^9.

Figure 6.2 indicates that our coreset construction method can efficiently

create coreset in different levels of input data size. Overall, the running time smoothly raises as the data size increases, proving that our coreset implementation is efficient. Noted that the slope of the construction time curve in Figure 6.2 suddenly become steep when the points size is around 10^8. We regard this as a normal situation since the input data size raises to five times from 10^8 to $5 \cdot 10^8$, which leads the time spend from roughly 50s to 250s. Perhaps more iteration (currently five times) of the same experiment can lead to a more precise calculation of program operating time. Overall, we can conclude that the coreset construction time increases linearly with the increase of data points.

6.4 Evaluate with Oversized Data

The merge-and-reduce version results in oversized data (data size is over than the GPU memory) is the most significant result of our project since the main motivation for us to carry on this project is to clarify whether the coreset method can exceed the GPU memory size limitation when dealing with big data. In this experiment, we set the coreset size to $N = 1000$, clustering size to $k = 5$.

First, we conduct the big data experiment to evaluate whether CuML can run when the input data is greater than the GPU memory size. We operate this experiment by Nvidia Tesla P100 card, with the $16GB$ GPU memory and Pascal architecture. See more information about the information of the GPU used in this experiment at Figure 6.3:

Our experiment results show that when CuML facing the super colossal data set, it will raise a **"cudaErrorMemoryAllocation out of memory"** error. And it is worth noting that we found CuML even can not deal with the data size of around $5.6G$ in CSV form with a $16G$ GPU memory. This $5.6G$ CSV file is the osm data file for Germany, which has 333418873 points in total. The running errors can be seen at Figure 6.4.

As a contrast, our merge-and-reduce coreset implementation can handle this size level data set. It cost **101.01** seconds to construct the coreset in the Germany osm data file that we described above. Furthermore, we test the merge-and-reduce implementation with our biggest data set, which combines nine European osm files with 1156442555 points in total. The result shows the merge-and-reduce version coreset method can successfully process this data set and generate a coreset in only a few minutes (**393.82** seconds, to be precise). And just as we presented above, our implementation can deal with any size of data since it read data in a stream and invoke the coreset computation chunk by chunk.

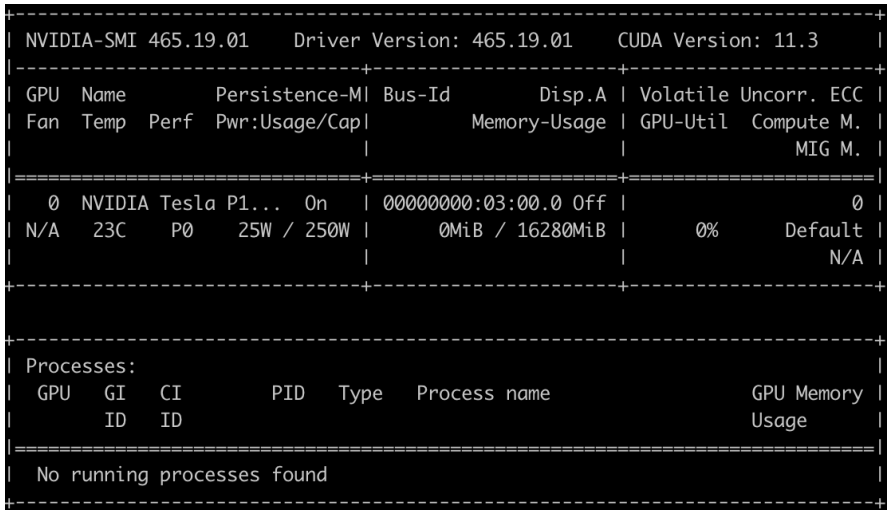

Figure 6.3: Nvidia Tesla P100 Card Information, which is the GPU card we used in this experiment

.

Therefore, compared to CuMl, our implementation breakthrough the limitation of GPU memory and can successfully cope with any size level data set in a relatively short time consuming even with a small machine. We regard this as the most impressive result and the most important contribution of this thesis project since it has a very practical significance in dealing with big data and machine learning problems.

```
Calling kmeans fit function!
Traceback (most recent call last):
  File "cuda_kmeans.py", line 124, in <module>
    cuml_kmeans_csv('./data/germany-latest.csv', 5, index=False)
  File "cuda_kmeans.py", line 66, in cuml_kmeans_csv
    kmeans_float.fit(data, sample_weight=weights)
  File "/home/wux4/.conda/envs/rapidsai/lib/python3.7/site-packages/cuml/interna
ls/api_decorators.py", line 410, in inner_with_setters
    return func(*args, **kwargs)
  File "cuml/cluster/kmeans.pyx", line 387, in cuml.cluster.kmeans.KMeans.fit
MemoryError: std::bad_alloc: CUDA error at: /home/wux4/.conda/envs/rapidsai/incl
ude/rmm/mr/device/cuda_memory_resource.hpp:69: cudaErrorMemoryAllocation out of
memory
```

Figure 6.4: CuML "cudaErrorMemoryAllocation out of memory" Error Information

.

Chapter 7

Discussion

As described in the Chapter 6, our coreset method can generate an accurate result in the k-means problem with a relatively lightly time and space cost. In addition, our approach breaks through the limitation of GPU memory, which is a severe shortcoming of CuML. Therefore, the evaluation results in implementing the coreset fully met the expectations we planned before starting the project.

Future works can be extended in multiple directions. First of all, our latest GPU implementation of the coreset method benefit from computing in parallel with multiple threads. Still, we have not optimized our code by many programming optimization strategies, for instance: **a. reuse data in registers, b. reuse data in shared memory, c. linear reading data.** Due to there is a bunch of vector computations in the process of coreset calculation; we are pretty sure after introducing these technologies, our coreset construction method will be more fast and efficient.

Furthermore, we evaluate the performance of coreset with k-means problems in this thesis project. Future works may extend to assess the accuracy of coreset in dealing with other clustering problems, like k-median, and other machine learning problems like regression, PCA, etc. Just as we mentioned above, there is already much theoretical research related to constructing coreset for different machine learning problems [13, 35]. Consequently, future implementation works may be conducted upon those academic research.

In terms of providing users (which may include coreset researchers, students and engineers) open-source software for efficiently building coresets, we may package our programs into python libraries in the future. Consequently, this work will increase the actual value of our implementation and prevent users from straying in our CPU code and spend considerable time establishing the operating environment.

Chapter 8

Conclusions

This thesis project aims to efficiently implement the modern coreset computation frameworks for dealing with k-means clustering problems. The primary motivation is to evaluate the practical performance of coreset in k-means problem and benchmark against the well-used CuML framework. As you can see in the Chapter 6, the results indicate our coreset construction method can build an informative coreset to obtain an accurate clustering result in solving the k-means problem. We compare the result of our coreset method with the popular CuML framework, and the contrast shows that even with a very small coreset size ($N = 100$), the clustering result generated from coreset can under 5 percent of relative errors with the result computed from the original massive data set by CuML. Furthermore, our results indicate with a specific range of coreset size ($[800, 20000]$), the clustering centers in the coreset can even have less objective values comparing to the clustering centers in the original data set, which is very impressive.

As for the generation speed, our method can process the data over $\mathbf{10^9}$ points in a few minutes. In contrast, CuML occurs a 'Run out of GPU memory error' when coping with the data that has roughly $\mathbf{3 \cdot 10^8}$ points. Due to the stream data reading and chunk by chunk computing property of the merge-and-reduce implementation, our coreset method can robust process any level of input data size in a common GPU machine by appropriate parameter setting. In conclusion, this proves that our coreset implementation breaks through the GPU memory restriction in processing the big data, which has an extraordinary practical meaning and usability.

Finally, our future directions may include speed up our GPU computation by utilizing the state of the art optimizing strategies; extend the implementation works to other machine learning problems, such as k-median; encapsulate our program as a Python library for users to simply and efficiently utilizing our method to construct coreset.

Bibliography

[1] Agarwal, P. K., Har-Peled, S., Varadarajan, K. R., et al. Geometric approximation via coresets. *Combinatorial and computational geometry 52* (2005), 1–30.

[2] Agarwal, P. K., Kumar, N., Sintos, S., and Suri, S. Efficient algorithms for k-regret minimizing sets. *arXiv preprint arXiv:1702.01446* (2017).

[3] Arthur, D., and Vassilvitskii, S. k-means++: The advantages of careful seeding. Tech. rep., Stanford, 2006.

[4] Bādoiu, M., Har-Peled, S., and Indyk, P. Approximate clustering via core-sets. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing* (2002), pp. 250–257.

[5] Balcan, M.-F. F., Ehrlich, S., and Liang, Y. Distributed k-means and k-median clustering on general topologies. *Advances in Neural Information Processing Systems 26* (2013), 1995–2003.

[6] Braverman, V., Jiang, S. H.-C., Krauthgamer, R., and Wu, X. Coresets for clustering in excluded-minor graphs and beyond. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2021), SIAM, pp. 2679–2696.

[7] Campbell, T., and Broderick, T. Bayesian coreset construction via greedy iterative geodesic ascent. In *International Conference on Machine Learning* (2018), PMLR, pp. 698–706.

[8] Data, I. W. I. B. Bring big data to the enterprise, 2012.

[9] Ding, H., and Xu, J. A unified framework for clustering constrained data without locality property. *Algorithmica 82*, 4 (2020), 808–852.

[10] Drineas, P., Mahoney, M. W., and Muthukrishnan, S. Sampling algorithms for l 2 regression and applications. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm* (2006), pp. 1127–1136.

[11] Epstein, D., and Feldman, D. Quadcopter tracks quadcopter via real-time shape fitting. *IEEE Robotics and Automation Letters 3*, 1 (2017), 544–550.

[12] Feldman, D. Core-sets: Updated survey. *Sampling Techniques for Supervised or Unsupervised Tasks* (2020), 23–44.

[13] Feldman, D., Fiat, A., and Sharir, M. Coresets forweighted facilities and their applications. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)* (2006), IEEE, pp. 315–324.

[14] Feldman, D., and Langberg, M. A unified framework for approximating and clustering data. In *Proceedings of the forty-third annual ACM symposium on Theory of computing* (2011), pp. 569–578.

[15] Feldman, D., Schmidt, M., and Sohler, C. Turning big data into tiny data: Constant-size coresets for k-means, pca, and projective clustering. *SIAM Journal on Computing 49*, 3 (2020), 601–657.

[16] Frahling, G., and Sohler, C. A fast k-means implementation using coresets. *International Journal of Computational Geometry & Applications 18*, 06 (2008), 605–625.

[17] Funke, S., and Laue, S. Bounded-hop energy-efficient broadcast in low-dimensional metrics via coresets. In *Annual symposium on theoretical aspects of computer science* (2007), Springer, pp. 272–283.

[18] Garland, M., Le Grand, S., Nickolls, J., Anderson, J., Hardwick, J., Morton, S., Phillips, E., Zhang, Y., and Volkov, V. Parallel computing experiences with cuda. *IEEE micro 28*, 4 (2008), 13–27.

[19] Gaudeul, A. Do open source developers respond to competition? the latex case study. *Review of Network Economics 6*, 2 (2007).

[20] Har-Peled, S. A simple algorithm for maximum margin classification, revisited. *arXiv preprint arXiv:1507.01563* (2015).

[21] Har-Peled, S., and Kushal, A. Smaller coresets for k-median and k-means clustering. *Discrete & Computational Geometry 37*, 1 (2007), 3–19.

[22] Har-Peled, S., and Mazumdar, S. On coresets for k-means and k-median clustering. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing* (2004), pp. 291–300.

[23] Hellerstein, J. Parallel programming in the age of big data. *Gigaom Blog 4*, 4 (2008), 4.

[24] Henzinger, M., and Kale, S. Fully-dynamic coresets. *arXiv preprint arXiv:2004.14891* (2020).

[25] Huang, L., and Vishnoi, N. K. Coresets for clustering in euclidean spaces: Importance sampling is nearly optimal. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing* (2020), pp. 1416–1429.

[26] Khan, M., Wu, X., Xu, X., and Dou, W. Big data challenges and opportunities in the hype of industry 4.0. In *2017 IEEE International Conference on Communications (ICC)* (2017), IEEE, pp. 1–6.

[27] Krishna, K., and Murty, M. N. Genetic k-means algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 29*, 3 (1999), 433–439.

[28] Kuhlman, D. *A python book: Beginning python, advanced python, and python exercises*. Dave Kuhlman Lutz, 2009.

[29] Langberg, M., and Schulman, L. Universal epsilon-approximators for integrals. pp. 598–607.

[30] Lee, S. S., Won, D., and McLeod, D. Discovering relationships among tags and geotags. In *ICWSM* (2008).

[31] Liang, Y., Balcan, M.-F., and Kanchanapally, V. Distributed pca and k-means clustering. In *The Big Learning Workshop at NIPS* (2013), vol. 2013, Citeseer.

[32] Lloyd, S. Least squares quantization in pcm. *IEEE transactions on information theory 28*, 2 (1982), 129–137.

[33] Lu, H., Li, M.-J., He, T., Wang, S., Narayanan, V., and Chan, K. S. Robust coreset construction for distributed machine learning. *IEEE Journal on Selected Areas in Communications 38*, 10 (2020), 2400–2417.

[34] Lucic, M., Faulkner, M., Krause, A., and Feldman, D. Training gaussian mixture models at scale via coresets. *The Journal of Machine Learning Research 18*, 1 (2017), 5885–5909.

[35] Maalouf, A., Jubran, I., and Feldman, D. Fast and accurate least-mean-squares solvers. *arXiv preprint arXiv:1906.04705* (2019).

[36] MacQueen, J., et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability* (1967), vol. 1, Oakland, CA, USA, pp. 281–297.

[37] Mahabadi, S., Indyk, P., Gharan, S. O., and Rezaei, A. Composable core-sets for determinant maximization: A simple near-optimal algorithm. In *International Conference on Machine Learning* (2019), PMLR, pp. 4254–4263.

[38] Mahajan, M., Nimbhorkar, P., and Varadarajan, K. R. The planar k-means problem is np-hard. *Theor. Comput. Sci. 442* (2012), 13–21.

[39] Meek, C., Thiesson, B., and Heckerman, D. The learning curve method applied to clustering. In *AISTATS* (2001).

[40] Molina, A., Munteanu, A., and Kersting, K. Coresets for dependency networks. *arXiv preprint arXiv:1710.03285* (2017).

[41] Mooney, P., Minghini, M., et al. A review of openstreetmap data.

[42] Munteanu, A., and Schwiegelshohn, C. Coresets-methods and history: A theoreticians design pattern for approximation and streaming algorithms. *KI-Künstliche Intelligenz 32*, 1 (2018), 37–53.

[43] Narayanan, S., and Nelson, J. Optimal terminal dimensionality reduction in euclidean space. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing* (2019), pp. 1064–1069.

[44] Nolet, C. J., Lafargue, V., Raff, E., Nanditale, T., Oates, T., Zedlewski, J., and Patterson, J. Bringing umap closer to the

speed of light with gpu acceleration. *arXiv preprint arXiv:2008.00325* (2020).

[45] Oliphant, T. E. Python for scientific computing. *Computing in Science & Engineering 9*, 3 (2007), 10–20.

[46] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . `https://www.openstreetmap.org`, 2017.

[47] Oussous, A., Benjelloun, F.-Z., Lahcen, A. A., and Belfkih, S. Big data technologies: A survey. *Journal of King Saud University-Computer and Information Sciences 30*, 4 (2018), 431–448.

[48] Owens, J. D., Houston, M., Luebke, D., Green, S., Stone, J. E., and Phillips, J. C. Gpu computing. *Proceedings of the IEEE 96*, 5 (2008), 879–899.

[49] Oyelade, O., Oladipupo, O. O., and Obagbuwa, I. Application of k means clustering algorithm for prediction of students academic performance. *arXiv preprint arXiv:1002.2425* (2010).

[50] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research 12* (2011), 2825–2830.

[51] Pelleg, D., and Moore, A. Accelerating exact k-means algorithms with geometric reasoning. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining* (1999), pp. 277–281.

[52] Pfefferbaum, B., and North, C. S. Mental health and the covid-19 pandemic. *New England Journal of Medicine 383*, 6 (2020), 510–512.

[53] Phillips, J. M., and Tai, W. M. Near-optimal coresets of kernel density estimates. *Discrete & Computational Geometry 63*, 4 (2020), 867–887.

[54] Schmidt, M., Schwiegelshohn, C., and Sohler, C. Fair coresets and streaming algorithms for fair k-means. In *International Workshop on Approximation and Online Algorithms* (2019), Springer, pp. 232–251.

[55] Stisen, A., Blunck, H., Bhattacharya, S., Prentow, T., KjÃ¦rgaard, M., Dey, A., Sonne, T., and Jensen, M. Smart devices are different: Assessing and mitigatingmobile sensing heterogeneities for activity recognition. pp. 127–140.

[56] Stroustrup, B. *The C++ programming language*. Pearson Education India, 2000.

[57] Stroustrup, B., and December, B. Bjarne stroustrup.

[58] Sun, S., Cao, Z., Zhu, H., and Zhao, J. A survey of optimization methods from a machine learning perspective. *IEEE transactions on cybernetics 50*, 8 (2019), 3668–3681.

[59] Theis, T. N., and Wong, H.-S. P. The end of moore's law: A new beginning for information technology. *Computing in Science & Engineering 19*, 2 (2017), 41–50.